VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2016 Tomáš Přívozník

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Přívozník**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Daniel Stříbný**

Konzultant bakalářské práce: Ing. Radoslav Glajc

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 21. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 21. dubna 2016

.....

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

Rád bych na tomto místě poděkoval společnosti Tieto Czech s.r.o. za možnost vykonání odborné praxe a také lidem z týmu pracujících na Tieto Share 2.0.

V neposlední řadě bych chtěl poděkovat svému vedoucímu bakalářské práce Ing. Danielovi Stříbnému za připomínky a vedení této práce.

Abstrakt

Tato bakalářská práce popisuje průběh mé odborné praxe a rozebírá mnou uplatněné dovednosti během této praxe. Možnost absolvování odborné praxe mi poskytla firma Tieto Czech s.r.o. na pozici Software Developer.

Na začátku práce je představení firmy a mé pracovní zařazení v průběhu praxe. V další kapitole je popis projektů, na kterých jsem pracoval a seznam mých úkolů. Následující kapitola se věnuje popisu řešení jednotlivých úkolů. Závěr práce obsahuje popis mnou uplatněných znalostí a dovedností. Poslední část práce se věnuje celkovému zhodnocení odborné praxe a dosažených výsledků.

Klíčová slova: odborná praxe, Tieto, Java, Spring Framework, Alfresco, JavaScript

Abstract

This bachelor thesis describes process of my professional practice and evaluation of use of mine skills during this practice. Company Tieto Czech s.r.o. gave me a posibility to do professional practice on the position of Software Developer.

At the beginning of the thesis there is an introduction of the company and my assignment during the practice. In the next chapter there is description of the projects that I worked on and the list of my tasks. The following chapter is dedicated to the description of solution of the particular tasks. Conclusion of thesis contains description of the knowledge and skills used by me. The last part of the thesis is devoted to the overall evaluation of my professional practice and achievements.

Key Words: professional practice, Tieto, Java, Spring Framework, Alfresco, JavaScript

Obsah

Seznam použitých zkratek a symbolů

AMP	–	Alfresco Module Package
API	–	Application Programming Interface
CMIS	–	Content Management Interoperability Services
CSS	–	Cascading Style Sheets
DAO	–	Data Access Object
ECM	–	Enterprise Content Management
FTL	–	FreeMarker Template Language
HR	–	Human resources
IT	–	Information technology
JSF	–	JavaServer Faces
JSON	–	JavaScript Object Notation
LDAP	–	Lightweight Directory Access Protocol
MVC	–	Model-view-controller
OOP	–	Object Oriented Programming
ORM	–	Object Relational Mapping
REST	–	Representational state transfer
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
XML	–	EXtensible Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Hlavním důvodem proč jsem se rozhodl absolvovat odbornou praxi, bylo získání praktických zkušeností k teoretickému základu získanému po dobu studia. Také mě velice zajímalo, jak probíhá celý proces vývoje softwaru ve firemním prostředí. Skrze odbornou praxi jsem měl možnost si vyzkoušet práci v několika týmech, díky toho jsem mohl pracovat s kolegy ze zahraničí (například z Finska nebo Indie).

Ve firmě jsem pracoval na pozici Software Developer. V první části praxe jsem se podílel na vývoji webových aplikací od návrhu až po předání. V druhé části praxe jsem vytvářel novou a upravoval stávající funkcionalitu u projektu, který už byl v průběhu vývoje. Všechny tyto aplikace byly vyvinuty v jazyce Java s využitím Spring Frameworku.

V této práci popisuji projekty, na kterých jsem pracoval, s tím spojené úkoly a jejich řešení. V závěru popisuji, kterých znalostí jsem využil v průběhu praxe. Na konci této práce se věnuji dosaženým výsledkům a celkovému zhodnocení praxe.

2 Představení firmy

Společnost nese název Tieto Corporation od roku 2009. Předtím se společnost jmenovala TietoEnator po spojení finské korporace Tieto a švédské společnosti Enator. Finská společnost Tieto s prvotním názvem Tietotehdas Oy byla založena v roce 1968 ve finském městě Espoo. Společnost Enator vznikla v roce 1995 fúzí s firmou Celsius a.s. [1].

Tieto je největším dodavatelem IT služeb pro soukromý i veřejný sektor ve Skandinávii. Do České republiky společnost vstoupila v roce 2001 a v roce 2004 se Ostrava stala softwarovým centrem společnosti u nás. Tím se společnost stala jedním z největších zaměstnavatelů v oblasti IT v České republice [2]. Mezi hlavní poskytované služby patří:

- Aplikační outsourcing
- Business Information Exchange
- Business Intelligence
- Business a IT transformace
- Podnikové systémy
- Portálová řešení
- Vývoj a správa aplikací

2.0.1 Pracovní zařazení

Na odbornou praxi jsem se hlásil na pozici Java Developer. Abych mohl nastoupit na tuto pozici, musel jsem projít přijímacím řízením.

Přijímací řízení mělo dvě kola pohovorů, na kterých jsem musel prokázat svoji znalost programovacího jazyka Java a také požadovanou úroveň znalosti anglického jazyka. V první kole jsem se setkal s personální pracovnicí a v druhém kole přijímacího řízení pak s manažerem týmu.

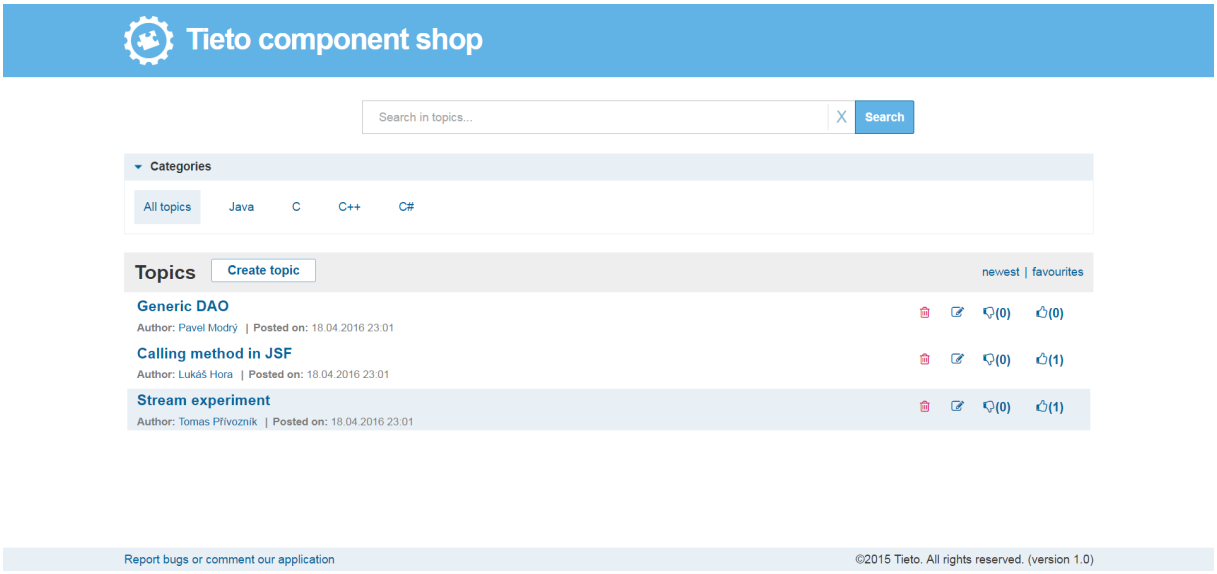
První dva pracovní dny jsem měl školení ohledně informací o společnosti, firemní politice a mnoha dalších věcech. Poté jsem začal pracovat na své pozici dva až tři dny v týdnu. Krátce po nastoupení jsem se účastnil školení, která mě seznámila s technologiemi používanými při vývoji webových aplikací. Pracoval jsem v týmech, které vyvíjely webové aplikace úplně od začátku, nebo rozšiřovali a upravovali již vytvořené aplikace. Při vývoji aplikací jsme provozovali agilní metodiku Scrum.

3 Zadání úkolů

3.1 Tieto component shop

Prvním projektem, na kterém jsem pracoval, byla webová aplikace pro interní použití. Úkolem bylo navrhnout a implementovat aplikaci, která měla sloužit jako úložiště zajímavých řešení ohledně programování. Jednotlivá řešení byla reprezentována uživatelům jako příspěvky na fóru (Obrázek 1: Tieto component shop - hlavní stránka). Uživatelé se do aplikace přihlašovali přes firemní LDAP. Normální uživatelé mohli přidávat, hodnotit a komentovat příspěvky, administrátoři mohli navíc spravovat kategorie a přidávat nebo odebírat administrátory. Příspěvky byly řazené do kategorií, podle kterých se dalo vyhledávat. Příspěvek obsahoval krátký název, popis a přílohu, což mohl být například zdrojový kód nebo obrázek. Uživatelé mohli příspěvek ohodnotit kladnou nebo zápornou zpětnou vazbou a také mohli k řešení přidat komentář.

Kromě chování aplikace uvedeného výše, jsme dostali zadáno, že tato aplikace má být vyvinuta v jazyce Java s využitím Spring Frameworku a pro vytvoření webových stránek jsme měli použít PrimeFaces Framework.



Obrázek 1: Tieto component shop - hlavní stránka

3.1.1 Klientská strana aplikace

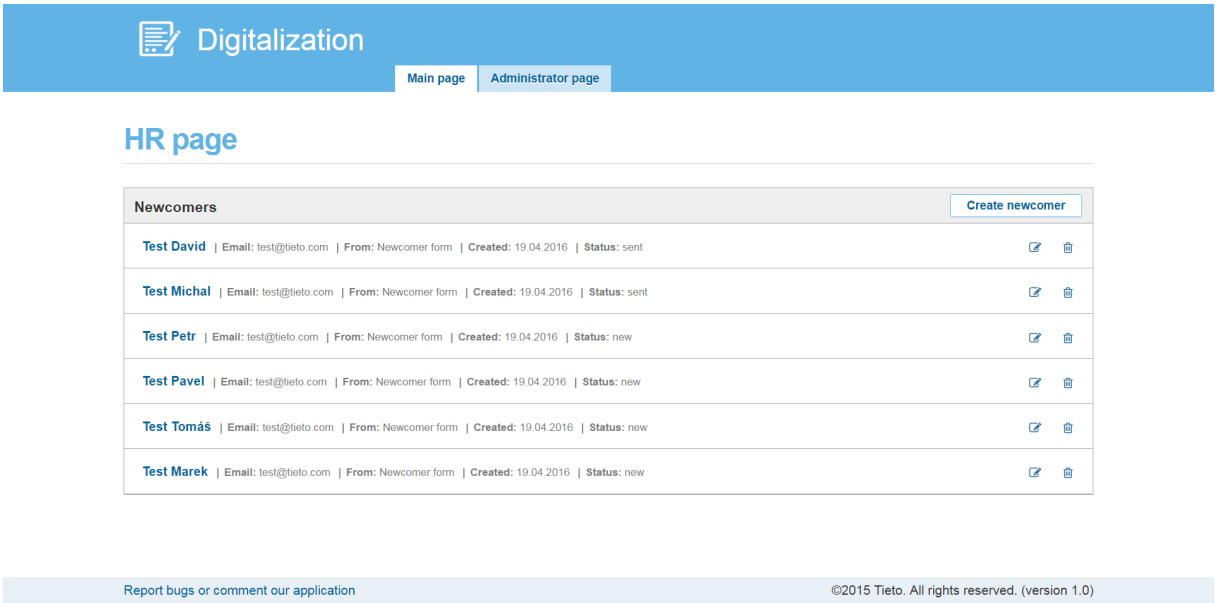
Na tomto projektu jsem z hlavní části vytvářel vzhled aplikace. To zahrnovalo vytváření JSF stránek a úpravu CSS souboru. Protože je klientská část aplikace úzce spojena se serverovou částí, vyzkoušel jsem si vytvoření nové nebo úpravu stávající funkcionality stránek. Vytváření klientské strany aplikace mi zabralo dvacet dnů.

3.2 Digitalization

Dalším projektem byla webová aplikace pro HR oddělení. Cílem aplikace bylo převést celý proces vyplnění a zpracování formuláře osobních informací nového zaměstnance (co nastupuje do Tieta v Ostravě) do elektronické formy. Formulář musel být vytvořen v české a anglické verzi. Do aplikace měli mít přístup pracovníci z HR oddělení a nastupující zaměstnanci. Přihlášení HR pracovníků se provádělo přes firemní LDAP. Pracovníci mohli vytvářet a rušit účty nastupujícím zaměstnancům a přidávat nebo odebírat přístupy do aplikace jiným HR pracovníkům. U formulářů mohli přidat komentář pouze HR pracovníci a v souvislosti s tím mohli měnit stav formuláře jako například schváleno, vráceno a jiné. Nastupujícím zaměstnancům musí být vytvořeny přihlašovací údaje, protože v době vyplňování formuláře ještě nemají vytvořené účty v Tieto doméně. Nastupující zaměstnanci byli zobrazeni na hlavní stránce od HR (Obrázek 2: Digitalization - hlavní stránka). Aplikace měla posílat e-maily v případě změny stavu formuláře, dále měla být schopná převést vyplněný formulář do tisknutelné podoby a odeslat jej k vytisknutí.

Toto byla první fáze projektu, v druhé fázi měly být přidány další formuláře (například formulář pro mateřskou dovolenou a další). Tyto nové formuláře by byly pro všechny zaměstnance, takže každý zaměstnanec by měl v aplikaci svůj vlastní účet. Uživatel by viděl formuláře, co vyplnil a formuláře co může vyplnit a odeslat.

Tuto aplikaci jsme také dělali v jazyce Java s využitím Spring Frameworku, pro vytvoření webových stránek jsme opět z největší části použili PrimeFaces Framework.



Obrázek 2: Digitalization - hlavní stránka

3.2.1 Návrh databázového modelu

Začal jsem návrhem databázového modelu, protože toto je podle mého názoru klíčová věc. Při návrhu jsem musel brát v úvahu pozdější rozšíření aplikace (například přidání nových formulářů). Navrhováním databázového modelu jsem strávil čtrnáct dní.

3.2.2 Vytvoření tříd databázového modelu

Po navržení databázového modelu jsem začal vytvářet v aplikaci třídy odpovídající tomuto modelu. Vytváření těchto tříd mi zabralo šest dnů.

3.2.3 Vytvoření třídy pro testovací data

Kvůli neustálému vytváření databáze při spuštění aplikace, jsem musel vytvořit třídu, která se zavolala po vytvoření databáze a naplnila ji testovacími daty, takže jsme nemuseli zadávat data ručně. Tuto třídu jsem vytvářel jeden den.

3.2.4 Vytvoření vrstvy DAO

Dalším úkolem bylo vytvoření přístupu do databáze, na tomto jsem pracoval jedenáct dnů. Pro přístup do databáze jsem použil DAO pro jednotlivé třídy.

3.2.5 Vytvoření servisní vrstvy

Pokračoval jsem vytvořením servisní vrstvy aplikace. Pro použití servisní vrstvy jsem se rozhodl kvůli lepšímu zabezpečení a pozdějšímu rozšíření aplikace. Servisní vrstvu jsem vytvořil za osm dní.

3.2.6 Vytvoření funkcionality pro stránky

Obsahem dalšího mého úkolu bylo vytvoření funkcionality pro stránky. Vytvářel jsem třídy obsahující metody provádějící akce v aplikaci, volané z JSF stránek. Vytvoření funkcionality mi trvalo dvacet dva dnů. Seznam vytvářených funkcionalit:

- Funkcionalita pro administrátory
- Funkcionalita pro komentáře
- Funkcionalita záložek na stránkách
- Funkcionalita pro nastupující zaměstnance
- Funkcionalita formuláře

3.2.7 Vytvoření zabezpečení aplikace

Velice důležitým úkolem bylo vytvoření zabezpečení naší aplikace. Spring obsahuje Spring Security Framework, který umožňuje omezení vstupu uživatele na určité stránky podle jeho role nebo autentifikaci přes různé systémy jako je LDAP [3]. Na zabezpečení aplikace jsem pracoval pět dnů.

3.3 Tieto Share 2.0

Dalším projektem, byla platforma poskytující řešení pro správu elektronického obsahu. Projekt je vytvořen na vývojové platformě Alfresco. Alfresco je open source ECM systém umožňující správu velkého množství digitálního obsahu [4]. Platforma Alfresco je vyvinuta v jazyce Java za využití Spring Frameworku. Byl jsem členem týmu vytvářející novou a upravující stávající funkcionalitu tohoto projektu.

3.3.1 Úprava stránek přes REST API

Uživatelé mohou vytvářet vlastní webové stránky, které mohou upravovat. Jednou z úprav je přidání Wiki stránek. Já jsem měl za úkol zjistit, jak se dají tyto stránky vytvořit, upravit a smazat s použitím REST API. Na toto jsem používal program Postman, který umožňuje posílat HTTP požadavky s různými metodami. Řešením těchto úkolů jsem strávil dva dny.

3.3.2 Odstranění skupiny EVERYONE z vyhledávání

V dalším úkolu jsem řešil odstranění skupiny EVERYONE z vyhledávání pro delegování práv určité složky. Tato skupina je v podstatě fiktivní a každý uživatel je v této skupině, proto musela být odstraněna z vyhledávání. Tím se znemožnilo přidání jakýchkoliv práv pro tuto skupinu. Tento úkol jsem řešil pět dnů.

3.3.3 Úprava projektu

Následujícím úkolem bylo přepsání hotového projektu vytvořeného v jazyce Java a přidání nové funkcionality. Hlavními důvody pro vytvoření projektu byly zjednodušení a zefektivnění práce s Alfresco systémem.

V projektu bylo vytvořeno přihlášení k webovému klientovi Alfresco aplikace zvanému Share a vytvoření stránky. Mým úkolem bylo zpřehlednit implementaci vytvoření stránky a naimplementovat smazání stránky, vytvoření složky a nahrání souboru. Na tomto úkolu jsem pracoval dvanáct dní.

3.3.4 Vložení JavaScript konzole do aplikace

Ještě předtím než jsem začal vytvářet Web Scripty, jsem přidal do svého lokálního projektu JavaScript konzoli, která umožňuje spouštět JavaScript kód nad Alfresco repository za běhu

instance. Výhodou této konzole je možnost testování JavaScript kódu a tím urychlení vývoje softwaru.

Mohl jsem začít vytvářením Web Scriptu přímo, ale to by bylo z počátku velice neefektivní oproti prvotnímu vytvoření funkcionality Web Scriptu v JavaScript konzoli. Přidání toho rozšíření do aplikace mi zabralo jeden den.

3.3.5 Web Script využití stránek

V dalším úkolu jsem vytvářel Web Script pro zjištění využití webových stránek v Alfresco Share. Využití se určuje spočítáním všech složek a souborů pod danou stránkou. Tento Web Script je používán ke zjištění využití platformy Tieto Share 2.0 od začátku nasazení do produkce. Web Script jsem vytvářel čtyři dny.

Web Script je služba, která reaguje na HTTP metodu jako je GET, POST, PUT a DELETE k provedení akce v Afrescu. Skládá se ze tří souborů. Nejdůležitějším je JavaScript soubor, obsahující funkcionalitu. Další je XML soubor obsahující vlastnosti Web Scriptu uzavřené v XML značkách specifikující jeho jméno, popis, URL a typ uživatele, který může Web Script spustit. Posledním souborem je FreeMarker šablona určující jak má vypadat výstup Web Scriptu [5]. Šablona je psaná v FTL, umožňující dynamicky zobrazovat data. Funkcionalita šablony je přirovnávána ke vzoru MVC. Protože zvlášť určujeme jaké data se mají poslat do šablony a zvlášť určujeme jak mají vypadat [6].

3.3.6 JavaScript pro smazání uzlů

Dostal jsem zadáno vytvořit JavaScript kód pro smazání, veškerých uzlů pod složkou documentLibrary specifické stránky. V Alfrescu je v podstatě vše uloženo jako uzel (například složka, soubor, stránka a další). DocumentLibrary je kořenová složka uchovávající obsah pod stránkou jako jsou složky, soubory, odkazy na složky a další. Tento kód se primárně používá během migrace dat do Alfresco systému. Při migraci stránky může dojít k chybě (například při přenosu dat), potom je nutné spustit migraci znova. Pro opětovné spuštění je potřeba smazat obsah stránky nebo stránku samotnou (a tím smazat i její obsah). Protože stránka může být už nastavená, je lepší smazat pouze její obsah. Ke smazání obsahu se použije výše uvedený kód, protože obsah může být tvořen několika tisíci soubory. Tento úkol jsem vyřešil za jeden den.

3.3.7 Web Script pro změnu práv určité složky

Náplní následujícího úkolu bylo vytvoření Web Scriptu, který pro určitou složku změnil práva k prohlížení. Web Script byl vytvořen kvůli nutnosti upravení přístupů pro velké množství složek, které musely být přímým potomkem složky documentLibrary. Bylo třeba odstranit práva k prohlížení složky pro již zmíněnou fiktivní skupinu EVERYONE a přidat stejná práva pro interní skupinu. Web Script jsem vytvářel pět dnů.

4 Řešení zadaných úkolů

4.1 Úkoly na Tieto component shop

4.1.1 Klientská strana aplikace

Při vytváření stránek jsme použili JSF Framework, tento MVC framework je využíván ke zjednodušení vývoje uživatelského rozhraní. Součástí toho frameworku je komponenta Model, což v naší aplikaci jsou ManagedBeans. To jsou Java třídy označené anotací @ManagedBean. Tyto třídy uchovávají data, která se ukážou uživateli nebo obsahují metody, které provedou nějakou akci v informačním systému. V těchto třídách jsem přidával novou a upravoval stávající funkcionalitu (například jsem vytvářel funkcionalitu týkající se kategorií). Další součástí JSF Frameworku jsou JSF stránky řadící se do komponenty View. Do stránky lze naimportovat už vytvořené knihovny obsahující programové struktury, jako je tabulka, kterou jde následně použít ve stránce. Jedna z knihoven byla PrimeFaces, tuto knihovnu jsem používal nejvíce, protože obsahuje velké množství dobře vypadajících komponent, má dobrou dokumentaci, používá jí mnoho vývojářů a nevztahují se na ni licenční podmínky [7].

Na hlavní stránce aplikace byl seznam vytvořených kategorií (Obrázek 1: Tieto component shop - hlavní stránka). Seznam se rozbalil po rozkliknutí a kategorie se zobrazovaly zleva doprava po řádcích. Bylo třeba zvýraznit aktuálně vybranou kategorii změnou barvy jejího pozadí. Tento problém jsem vyřešil vložením ternárního operátoru do jména třídy z CSS souboru. Uvnitř operátoru byla volána metoda, která byla uložena v JavaBeaně a pro každou kategorii vracela true nebo false, na základě toho jestli je daná kategorie vybrána. Podle návratové hodnoty metody se rozhodlo, která třída z CSS souboru se má na danou kategorii aplikovat a tím změnit barvu pozadí.

```
<p:accordionPanel activeIndex="-1" styleClass="categories section">
   <p:tab title="Categories">
      <ui:repeat var="category" value="#{categoryBean.listOfCategories}">
         <div class="#{topicBean.selectedCategory(category)?'categorySelected':'
             category'}">
            <p:commandLink value="#{category.name}"
              action="#{topicBean.setSelectedCategory(category)}" update="@form
                  " />
         </div>
      </ui:repeat>
   </p:tab>
</p:accordionPanel>
```

Výpis 1: Tieto component shop - zobrazení kategorií

Protože aplikace měla vypadat podle standardu Tieta, museli jsme mít upravený vzhled stránek. Vzhled pro základní prvky (například nadpisy) jsme už měli z předchozího projektu. Přidával jsem vzhled komponentám, které byly ve výchozím vzhledu. Dále jsem upravoval stávají vzhled a opravoval jsem chyby, které vznikaly při mixování několika stylů. V celé aplikaci jsme chtěli mít jednoduchý design. Snažil jsem se co nejvíce vyhnout použití tabulek, protože to je uživatelsky nepřívětivé. Proto jakmile jsme někde zobrazovali data v seznamu, řádky nebyly ohraničené čárou, ale pouze aktivní položka v seznamu měla jinou barvu pozadí (Obrázek 1: Tieto component shop - hlavní stránka). Toto jsem udělal přes třídu v CSS souboru, která byla v JSF stránce přiřazena dané komponentě, vytvářející seznam.

Jedním z dalších problémů bylo odsazení obrázku v tlačítku. Aby byl obrázek uprostřed tlačítka, musela být vytvořená nová třída v CSS souboru s jiným odsazením, která byla nastavena pouze pro tlačítka v určité komponentě. Také jsem například řešil upravování velikostních poměrů oddílů v řádku seznamu při zvětšení stránky.

4.2 Úkoly na Digitalization

4.2.1 Návrh databázového modelu

S ohledem na pozdější rozšíření aplikace jsem se rozhodl udělat z tabulky Uživatel v podstatě hlavní tabulku. Při vytváření návrhů tabulek formuláře jsem zvažoval, jestli udělat jednu velkou tabulku pro formulář nebo formulář rozdělit na několik menších tabulek. Nakonec jsem se rozhodl pro více menších tabulek a ty jsem navázal na jednu tabulku, která zajišťovala celistvost formuláře, tato tabulka navíc obsahovala obecné informace o formuláři jako jeho stav a jazyk.

4.2.2 Vytvoření tříd databázového modelu

Třídy jsem označil anotacemi, na základě kterých Hibernate vytvořil databázi za nás při každém spuštění aplikace. To jestli se databáze smaže a vytvoří znova nebo zůstane nezměněna, záviselo na tom, co bylo nastavené ve vlastnostech našeho projektu. Při vývoji nám vyhovovalo a dokonce bylo žádoucí vytvoření nové databáze při každém spuštění, kvůli dodatečným úpravám databázového modelu.

Správu primárních klíčů tabulek jsem přenechal databázi, tím že jsem atribut id označil anotací pro automatické generování.

Vztahy mezi tabulkami jsou také vytvářeny přes anotace a pro každý typ vztahu je jiná anotace. U anotace je možné nastavit vlastnost určující unikátnost atributů pro daný sloupec nebo u vztahu mezi tabulkami jde specifikovat, podle jakého atributu se má vztah mapovat. Jednou z dalších věcí co lze ovlivnit, je chování vztahu. To je možné nastavením vlastnosti cascade, která určí jaká operace (například vytvoření nebo smazání) se má provést pro každého potomka daného objektu. Tato vlastnost může velice ulehčit práci programátora, ovšem tak samo ji může znepříjemnit. U jednoho vztahu jsem měl nastavenou vlastnost pro řetězové odstranění, fungující perfektně při smazání předka, ovšem když jsem chtěl smazat, některého z potomků, odstranění

se neprovedlo. Potomek nebyl odstraněn ani potom co jsem ho odstranil z kolekce předka. Nakonec jsem to vyřešil odstraněním vlastnosti kaskádového mazání a přidal jsem vlastnost pro odstranění prvku při zrušení vazby na předka.

4.2.3 Vytvoření třídy pro testovací data

Nejprve jsem vytvořil kolekce dat, pro zaplnění tabulek. Potom jsem přes servisní vrstvu nechal tyto kolekce vložit do databáze metodou hromadné operace. Aby se vytvořila instance třídy, musela být třída přidána do konfiguračního souboru ApplicationContext.

4.2.4 Vytvoření vrstvy DAO

V minulém projektu jsme použili přídavek Spring Roo, který sám vytvoří vrstvu pro přístup do databáze. Zkušenější zaměstnanci mi doporučili, abych se vyhnul použití Spring Roo, protože generuje velké množství tříd navíc. Někdy se vyskytly chyby ve vygenerovaných dotazech. Například chybělo AND v podmínce, takže jsme dostávali výjimky o chybných dotazech. Zjistili jsme, že je to známá chyba v aktuální verzi, kterou jsme používali.

I když jsem nepoužil Spring Roo, chtěl jsem použít stejný způsob přístupu do databáze jako v předchozím projektu, použitím EntityManageru a označením metod pro přístup do databáze anotací @Transactional. Bohužel mi tento přístup nefungoval, proto jsem vytvářel transakce sám přes EntityManagerFactory. Až jsem měl vytvořených několik tříd s EntityManagerFactory, zjistil jsem, proč mi nefungoval předchozí přístup. Bylo to způsobeno chybějící závislostí pro aspekty v konfiguračním souboru projektu. Zhruba ve stejný čas jsem se dozvěděl, že použitím anotace @Transactional nechávám zcela na EntityManageru, jestli se transakce provede nebo ne. Takže se může stát, že pokud zavolám hned po sobě několik metod označených touto anotací, EntityManager z toho může udělat jednu transakci, ale správně by tyto metody měly být jako oddělené transakce. Z toho důvodu jsem se rozhodl dále pokračovat s použitím EntityManagerFactory.

Všechny tyto třídy mají anotaci @Repository. Je to anotace používána Spring Frameworkem pro třídy přistupující do databáze. Po nastavení skenování komponent v konfiguračním souboru, se načtou všechny stereotypy (@Repository je jedním ze stereotypu) v aplikaci, to zajistí bezproblémové použití těchto tříd v projektu. Při vytváření DAO jsem vždy vytvořil rozhraní s metodami a třídu s implementací. V aplikaci jsem pracoval s rozhraním, abych přímo nevystavoval implementaci.

4.2.5 Vytvoření servisní vrstvy

Jedním z argumentů pro vytvoření servisní vrstvy bylo rozšíření aplikace, konkrétně přidání dalšího typu uživatele. Dalším argumentem bylo vzájemné propojení vrstvy DAO a servisní vrstvy, kdy DAO přistupuje do databáze a servisní vrstva posílá přijímá data z DAO, navíc může obsahovat logiku pro řízení transakcí. Pravděpodobně posledním pádným argumentem bylo zabezpečení, protože se nepracuje přímo s vrstvou, která přistupuje do databáze. Místo toho

se pracuje s jinou vrstvou, zaobalující vrstvu přistupující do databáze. Tím se zvýší bezpečnost přístupu do databáze.

Ke každé třídě jsem vytvářel rozhraní, abych při použití servisní vrstvy skryl její implementaci. Každá třída byla označená dvěma anotacemi. První byla @Service, která se používá pro třídy této vrstvy. Druhá anotace byla @Scope typu session. Tento typ určoval životnost jedné třídy pouze po dobu HTTP relace. Uvnitř tříd byly metody pro volání metod z DAO pro uložení nebo získání objektů. Metody také odchytávaly a následně zpracovávaly výjimky, které se mohly vyskytnout při pokusu uložení stejného objektu do databáze nebo jiném problému.

4.2.6 Vytvoření funkcionality pro stránky

Funkcionalita stránek je implementována v ManagedBeans, což jsou třídy patřící do MVC architektonického vzoru, o kterém jsem se zmínil už dříve. Tyto třídy jsem vytvářel s ohledem na přehlednost a znovupoužitelnost. Proto každá třída obsahovala funkčnost týkající se jednoho celku. Zároveň jsem se snažil vyhnout anti vzorům jako je The God Class. V podstatě jde o to, že objekt obsahuje příliš mnoho ne-li všechnu funkcionalitu. Potom se takový objekt stává nepřehledný a špatně se udržuje. Způsob jak se tomuto vyvarovat je rozdělit problém na menší části [8].

4.2.6.1 Funkcionalita pro administrátory Třída má metody pro vypsání, přidání a odstranění administrátorů. Při vytváření nového administrátora se nejdříve podle zadaného loginu zaměstnance zjistí jeho údaje z firemního LDAPu. Po úspěšném získání informací se vytvoří nový uživatel, s nastavenou rolí administrátora (uživatelé v aplikaci jsou rozlišování podle rolí uložených ve výčtovém typu). Uživatel se uloží do databáze a následně se aktualizuje seznam administrátorů. U odstraňování administrátorů je funkčnost nastavena tak, aby nebylo možné smazat úplně všechny administrátory, neboť pouze administrátoři mohou provést vytvoření a odstranění administrátorů.

4.2.6.2 Funkcionalita pro komentáře Třída pro správu komentářů obsahuje metody pro vytvoření, smazání a získání komentářů pro určitý formulář. Tady se uplatňuje znovupoužitelnost, protože se s komentáři provádějí stále stejné operace a mění se pouze formulář. Tím že je specifikováno, jakému formuláři se má operace přiřadit, lze toto řešení využít i v budoucnu, když budou přidány nové formuláře. U přidání komentáře se přidává jméno autora automaticky podle údajů přihlášeného uživatele.

4.2.6.3 Funkcionalita záložek na stránkách Třída obsahuje funkcionalitu pro záložky na stránkách. Administrátoři mají více záložek než noví zaměstnanci. Vytvořil jsem šablonu, která obsahovala komponentu tabMenu z knihovny PrimeFaces. Záložku, kterou měl uživatel aktuálně zobrazenou, se dalo rozeznat jinou barvou jejího pozadí. Toto se provádělo nastavením indexu pro aktuální zobrazenou záložku v komponentě tabMenu. Pro získávání aktuálního indexu jsem

vytvořil metodu, která byla volána při každém načtení stránky. Nejdříve jsem si zjistil URL adresu stránky, následně jsem zjistil pouze název stránky a podle toho jsem vrátil index záložky. Záložky pro oba typy uživatelů musely mít stejné indexy, protože i když komponenta má několik záložek, při načtení stránky dojde k odstranění několika záložek, na základě role uživatele.

Pro zobrazení různých záložek pro různé typy uživatelů, jsem využil zabezpečení Spring Security Frameworku. Každá záložka byla ohraničena XML značkami pro autorizaci, kde se specifikovala role uživatele, pro kterého má být záložka viditelná.

```
<p:tabMenu id="menuTab" activeIndex="#{menuBean.getActivePage()}">
   <sec:authorize access="hasRole('ROLE_NEWCOMER')">
      <p:menuitem id="userTab" value="Newcomer"
         action="#{menuBean.redirectPage(0)}" update="menuTab">
      </p:menuitem>
   </sec:authorize>
</p:tabMenu>
```

Výpis 2: Digitalization - zobrazení záložek

Pro přesměrování se zavolala metoda z příslušné Beany, která nedostala přímo název stránky, na kterou se má přesměrovat ale pouze její číselné označení a podle toho se ve funkci rozhodlo, na jakou stránku se provede přesměrování.

4.2.6.4 Funkcionalita pro nové zaměstnance Třída obsahuje metody pro získání všech nových zaměstnanců z databáze, uložení a smazání nového zaměstnance a generování jeho přihlašovacích údajů. Při vytváření nového zaměstnance bylo nutné nejdříve zadat jeho jméno, příjmení, e-mail a další atributy související s jeho formulářem. Po úspěšné validaci údajů se vygenerovalo jeho přihlašovací jméno a heslo. Přihlašovací jméno bylo kombinací prvních několika znaků ze jména a příjmení. Nejdříve byly řetězce se jménem a příjmením upraveny aby neobsahovaly bílé znaky. Poté musely být řetězce zbaveny speciálních znaků ve jméně, jako jsou písmena s háčkem, čárkou a mnoho jiných. Tato písmena musela být převedena na stejná písmena, bez diakritiky. To jsem provedl s pomocí třídy normalizující text a regulárního výrazu. Pro oba řetězce jsem provedl normalizaci a z jejich prvních několika znaků jsem vytvořil přihlašovací jméno. Při vytvoření nového zaměstnance se rovnou vytvoří i jeho formulář. U odstranění nového zaměstnance se uplatňuje kaskádové smazání, stačí smazat záznam o uživateli a jeho přihlašovací údaje a formulář se také odstraní.

4.2.6.5 Funkcionalita formuláře Tato třída obsahuje nejvíce funkcionality. Nejtěžší bylo vymyslet, jak se bude provádět přenášení formuláře mezi jednotlivými Beanami. Jedním ze způsobů bylo poslat zašifrované id formuláře jako parametr v URL adrese. Ale tomu jsem se chtěl vyhnout, protože takhle bych vystavil aplikaci potenciálnímu riziku, samozřejmě kromě zašifrovaného id formuláře by bylo přidáno další zabezpečení omezující přístup, použitím metod

Spring Security Frameworku. Nakonec jsem se rozhodl, že budu v JSF stránce volat metodu, do které pošlu uživatele (nového zaměstnance) a podle něj se načte formulář a dojde k přesměrování.

Při načítání formuláře se zjistí jaký je jeho stav. Pokud má stav nově vytvořeného formuláře, jsou vytvořeny nové instance tříd. Jejich atributy jsou mapovány na pole pro zadání textů ve formuláři. Tímto se vyčistí všechny pole ve formuláři. Pole obsahující úplně základní informace jako jméno a příjmení zůstanou vyplněné. Toto se provedlo, když byl formulář nově vytvořený, ale pokud není, zavolá se metoda pro načtení.

Uvnitř metody pro načtení formuláře je na začátku zjištěno, jestli je formulář v českém nebo anglickém jazyce. Podle toho se nastaví, v jakém jazyce se budou zobrazovat pole se seznamy atributů. Dalším krokem je čtení částí formuláře, což jsou tabulky v databázi. Následně přichází na řadu čtení komentářů. Nakonec dojde k přesměrování.

Formulář se skládá z několika částí, které jsou tvořené z atributů patřících do stejné skupiny. Některé skupiny můžou zůstat nevyplněné, k tomuto účelu jsou ve formuláři pole určující povinnost, kde je na výběr možnost ano nebo ne.

Při ukládání formuláře se rozlišuje na základě jeho stavu, jestli se provede metoda pro vytvoření nového formuláře nebo pro úpravu již existujícího. V podstatě obě metody jsou stejné (i databázové operace jsou stejné), rozdíl je v tom, že při vytváření nového formuláře se nemusí ošetřovat, jestli nemají být smazané některé nepovinné skupiny. Při odstranění nepovinné skupiny dojde k přerušení vazby mezi skupinou a formulářem. A následně při ukládání je skupina smazána.

Předchozí zaměstnání je část formuláře, která obsahuje několik polí. Předchozí zaměstnání se přidává a upravuje přes dialogové okno. Nechtěl jsem, aby se jakákoliv změna předchozího zaměstnání promítala hned do databáze. Problém nastal, když v databázi bylo několik záznamů a uživatel je na stránce smazal, vše se normálně uložilo a formulář neměl žádné předchozí zaměstnání po opětovném načtení. Bohužel předchozí zaměstnání zůstaly v databázi, pouze nebyly přiřazené formuláři. Vyřešil jsem to přidáním dalšího listu předchozích zaměstnání, jakmile uživatel smaže zaměstnání, odstraní se z původního listu a přesune se do nového listu pro smazané zaměstnání. Předtím než se formulář odešle k uložení, se všem odstraněným zaměstnáním smaže formulář. Potom jsou oba listy spojeny do jednoho. Může se zdát zbytečné mít dva listy, když nakonec odstraněným zaměstnáním jenom odstraním formulář, ale seznam zaměstnání je tvořen komponentou, která je mapovaná na list se zaměstnáním. Komponenta zobrazí vše co je v listu a není rozdíl mezi tím, jestli má nějaké zaměstnání smazaný formulář nebo ne. Při ukládání formuláře se projde celý list se zaměstnáními a smažou se zaměstnání, které nemají formulář. Nejdříve se zaměstnání vyhledá v databázi a potom se smaže, tak nedojde k vyvolání výjimky se smazáním neexistujícího záznamu.

U formuláře pro nové zaměstnance jsou pole jako například rodinný stav, zde mohl uživatel vybrat jednu z předdefinovaných možností. Tyto možnosti jsou významově stejné pro oba jazyky, problém byl v tom, jak uložit tyto možnosti do databáze a přitom je později co nejjednodušeji získat bez toho aniž bych musel danou možnost převádět na jeden nebo druhý jazyk. Dobré řešení

poskytl výčtový typ, kdy jsou tyto možnosti reprezentovány jako čísla (což je ideální pro uložení do databáze, protože nemusím řešit, jaký jazyk se použije pro uložení), ale zároveň obsahovaly český i anglický název. Každá možnost rodinného stavu měla dvě metody, které vracely název v jiném jazyce, v obou formulářích se zavolaly jiné metody a tím se zobrazily různé jazyky.

```
public enum FamilyStatus {
   STATUS_SINGLE("svobodny", "single");

   private final String cz;
   private final String en;

   private FamilyStatus(String cz, String en) {
      this.cz = cz;
      this.en = en;
   }
   public String getStatusCz() {
      return cz;
   }
   public String getStatusEn() {
      return en;
   }
}
```

Výpis 3: Digitalization - výčtový typ pro rodinný stav

4.2.7 Vytvoření zabezpečení aplikace

Základní konfiguraci jsem měl už vytvořenou z minulého projektu, ale tu jsem musel přepsat a přidat ověřování uživatele podle vlastní databáze. Tato autentifikace se používá u nových zaměstnanců. Vytvořil jsem metodu sloužící pro získání dat uživatele na základě přihlašovacího jména, které je získáno pomocí instance servisní vrstvy. Metoda vracela jeho přihlašovací jméno spolu s heslem, rolemi a atributy specifikující validnost uživatele. Následně jsem vytvářel metodu, která se zavolá po úspěšné autentifikaci. Metoda vracela URL adresu, kam se má uživatel přesměrovat, URL se měnilo na základě uživatelovy role. Zbývalo vytvoření třídy pro získání rolí uživatele při přihlašování přes firemní LDAP. Metoda získala uživatele na základě přihlašovacího jména. Teď metoda vrací pouze roli administrátora, protože nikdo jiný se zatím do aplikace přes LDAP přihlásit nemůže. Tyto třídy neznamenají nic, pokud nejsou správně nadefinované v XML souboru, určující konfiguraci zabezpečení. V tomto souboru se mimo jiné specifikuje, jaké role jsou potřebné pro zobrazení dané stránky.

4.3 Úkoly na Tieto Share 2.0

4.3.1 Úprava stránek přes REST API

Nejdříve jsem začal smazáním stránky. Aby došlo ke smazání Wiki stránek musela být zavolána správně sestavená URL adresa s HTTP metodou DELETE. V URL adrese musela být specifikována stránka, pod kterou se má provést smazání a Wiki stránka, která měla být smazána. Po odeslání požadavku na dané URL se zavolal Web Script, který provedl smazání.

Dále jsem pokračoval vytvořením nové Wiki stránky. Tady jsem musel posílat JSON objekt, ve kterém byl specifikovaný název stránky a další vlastnosti. Tentokrát musela být použita HTTP metoda POST. Adresa URL byla podobná jako u smazání, kvůli tomu sestavení správné URL adresy už nebylo tak obtížné. Nebylo jednoduché zjistit, co má být posláno v objektu pro vytvoření nové stránky. Protože když jsem zavolal nad existující stránkou HTTP metodu GET, tak jsem dostal všechny informace o Wiki stránce ve formátu JSON. Nyní stačilo získané informace přepsat a odeslat je pro vytvoření stránky. I s přepsaným objektem se mi nedařilo stránku vytvořit. Po prostudování implementace jsem zjistil, že problém byl v názvech atributů. Názvy atributů byly jiné než, ty které jsem dostal v informacích o stránce. Po vložení správných atributů jsem úspěšně vytvořil Wiki stránku.

Nakonec jsem se zaměřil na změnu stránky. Po předchozích úkolech jsem měl dostatečné množství vědomostí na rychlé vyřešení úkolu. Tady bylo třeba zjistit, které atributy musí být poslány v objektu, aby se stránka upravila. Po prostudování implementace jsem zjistil, že v objektu musí být minimálně aktuální verze stránky a její název.

4.3.2 Odstranění skupiny EVERYONE z vyhledávání

Začal jsem tím, že jsem provedl vyhledávání v krokovacím módu prohlížeče. Tím jsem zjistil jaké JavaScript soubory se při vyhledávání zavolaly. Poté jsem tyto soubory procházel a sledoval jsem, kudy výsledek vyhledávání prochází. Nakonec jsem našel funkci, odkud se získává výsledek vyhledávání. Zjistil jsem, že data jsou získána z Web Scriptu, jenže data obsahovala více věcí než jen vyhledané skupiny. Takže jsem musel hledat dále.

Po delší době krokování jsem našel funkci, která upravovala výsledek vyhledávání do pole vyhledaných skupin. V této funkci jsem byl schopen smazat nežádoucí skupinu. Toto ovšem bylo řešení na klientské straně, takže by se mohlo stát, že někdo by tento kód mohl upravit a způsobit v aplikaci problémy. K tomu by nemělo dojít, protože v konfiguračním souboru se dá nastavit, že je zakázané upravovat JavaScript soubory na klientské straně.

Proto jsem hledal jiný způsob jak vyřešit tento problém na serverové straně. A to mně dovedlo k již zmíněnému Web Scriptu, ze kterého byla data získána. Funkcionalitu Web Scriptu tvoří JavaScript soubor. Tento JavaScript kód provedl vyhledání skupin a nakonec vždy přidal skupinu EVERYONE. Protože jak jsem již zmínil dříve, EVERYONE je fiktivní skupina. Tady stačilo upravit kód, aby nežádoucí skupina nebyla přidána, tím byl problém vyřešen a navíc vše bylo skryto před uživatelem.

Ale protože jsme chtěli zachovat původní soubor nezměněný pro případ, že by stará funkcionalita byla opět žádoucí, rozhodli jsme se úpravu provést jinak než přímým přepsáním kódu. Výsledné pole skupin bylo předáno do root objektu, který je předán dále. Nová funkcionalita byla v novém souboru, ve kterém se přes root objekt dalo upravit předané pole. Provedlo se vyhledání nežádoucí skupiny a ta byla smazána. Poté bylo pole opět předáno do root objektu. V konfiguracích pro Web Scripty se určilo, že po zavolání prvního JavaScript souboru, provádějící vyhledávání se zavolal druhý soubor upravující výsledek vyhledávání. Takto se dají jednoduše spravovat úpravy Web Scriptů.

4.3.3 Úprava projektu

Při přepisování stávající implementace, jsem kód rozdělil do několika tříd, aby splňoval některé ze zásad OOP jako je zapouzdřenost a znovupoužitelnost. Jedna třída obsahovala metody pro práci se stránkou, jiná třída obsahovala metody pro HTTP přihlášení do aplikace a tak dále. Poté jsem přidal načítání souboru ve formátu JSON, který byl poslán přes HTTP API, a podle něj byla vytvořena stránka s danými vlastnostmi. Je lepší načítat soubor s vlastnostmi než mít metody pro vytvoření JSON objektu, protože se může vytvořit nový typ stránky. U souboru přepíšu typ a vše bude fungovat, ale u metody bych musel zasahovat do implementace.

4.3.3.1 Implementace smazání stránky v Javě Abych mohl smazat stránku, musel jsem se přihlásit k repository aplikace. Přihlášení k repository aplikace se ovšem provádí jinak než k Share. Proto jsem musel naimplementovat nové přihlášení k Alfrescu. Toto přihlášení používá Alfresco ticket, což je řetězec označující správnou autentifikaci. K obdržení řetězce je potřeba na specifickou URL adresu poslat ve streamu objekt obsahující přihlašovací údaje administrátora. Ze streamu obdržíme surová data, která musíme převést na JSON. Poté sestavíme URL adresu obsahující název stránky, kterou chceme smazat a nakonec vložíme Alfresco ticket. Na sestavenou URL adresu pošleme HTTP požadavek s metodou GET a tím se provede Web Script, který smaže stránku.

4.3.3.2 Implementace vytvoření složky v Javě Pokračoval jsem implementací vytvoření složky. Vyzkoušel jsem oba předchozí způsoby přihlášení, ale ani jeden nefungoval. Proto jsem musel najít absolutně odlišný způsob jak vytvořit složku.

Po interní diskusi jsem se dozvěděl, že mám použít OpenCMIS, CMIS je otevřený standart, umožňující různým systémů spravující obsah spolupracovat přes internet a OpenCMIS je kolekce Java knihoven, frameworků a nástrojů okolo CMIS [9]. Abych byl schopný použít OpenCMIS musel jsem si stáhnout tyto knihovny do úložiště svého vývojového prostředí.

Práce s OpenCMIS je velice jednoduchá. Provádění změn v Alfrescu, za použití OpenCMIS, je prováděno přes OpenCMIS relaci. Pro získání relace stačí zavolat metodu z API, do které je předána kolekce atributů obsahující přihlašovací údaje a další atributy. Po získání relace přichází na řadu sestavení řetězce se stromovou strukturou jeho předků. Následně podle cesty je získána

složka, která bude předkem nové složky. Nad předkem se zavolá metoda pro vytvoření nové složky, která očekává kolekci vlastností, obsahující minimálně název složky a typ složky.

4.3.3.3 Implementace nahrání souboru v Javě Přišla na řadu implementace nahrání souboru, která se také provádí přes relaci OpenCMIS. Získávání relace jsem měl už vytvořeno, takže jsem rovnou začal pracovat na nahrávání souboru. Postup se skoro nelišil od vytváření složky. Opět je sestavena cesta, kam se má soubor nahrát a vytvořena kolekce s vlastnostmi souboru. Poté stačilo vytvořit stream z nahrávaného souboru a předat jej spolu s vlastnostmi do metody pro nahrání souboru.

4.3.4 Vložení JavaScript konzole do aplikace

Takovéto komponenty jsou přidány do aplikace jako AMP soubory. AMP soubor je kolekce souborů například kódu, XML, obrázků a jiné, které rozšiřují funkcionalitu aplikace [10]. Tyto soubory musí být přidány do konfiguračního souboru projektu.

4.3.5 Web Script využití stránek

Jakmile jsem měl konzoli, ve které jsem byl schopný spustit Javascript kód, začal jsem vytvářet funkcionalitu Web Scriptu. Kód vytvořený v konzoli je odlišný od toho, co je použitý ve finální verzi, protože v konzoli nejsou argumenty z URL adresy nebo root objekt.

Funkcionalitu jsem vytvořil dvojím způsobem. Nejprve jsem vytvořil funkcionalitu přes rekurzi a potom přes SQL dotazy.

V obou případech jsem si na začátku získal všechny stránky nacházející se pod Alfresco Share pomocí SQL dotazu. Dotaz může být upraven podle parametrů, co zadá uživatel, aby se vyhledávaly stránky podle jejího typu anebo také jména. U dotazu bylo zapotřebí ošetřit stránkování, protože v nastavení projektu je specifikován maximální počet položek, co může dotaz vrátit. Pokud bude mít Alfresco Share více stránek než je hodnota limitující výsledek dotazu, musí se dotaz zavolat znovu se změněným stránkováním, aby navázal na dotaz předcházející.

Při použití rekurze se nad každou stránkou zavolá funkce vracející pole přímých potomků (neboli uzlů). Pak se zavolá funkce pro procházení všech potomků a počítání všech složek a souborů. Při procházení potomků se používá algoritmus Prohledávání do hloubky. Uvnitř funkce se zjistí, jestli je daný potomek složka nebo soubor a zvětší se počítadlo. Nakonec se provede rekurzivní zavolání funkce s polem potomků aktuálního uzlu, jak už napovídá algoritmus Prohledávání do hloubky.

U druhého způsobu se pro každou stránku zavolají dva dotazy. U obou dotazů muselo být řešeno stránkování. Jeden dotaz se provedl nad tabulkou pro soubory a druhý dotaz nad tabulkou pro složky. Proto, aby dotaz vrátil pouze složky nebo soubory patřící dané stránce, byla použita funkce IN_TREE, která očekává referenci na uzel (stránku). Tato funkce specifikuje, v jaké

stromové struktuře se má vyhledávat. Počet složek nebo souborů se určuje počtem položek ve výsledku dotazu.

Po dokončení průchodu stránkami se tyto informace vloží do JSON objektu a předají se do root objektu. Poté jsem vytvářel XML soubor s vlastnostmi Web Scriptu, kde jsem nastavil název a popis. Jako poslední věc jsem vytvářel FreeMarker šablonu. Šablona měla klasický JSON formát. Protože bylo možné zobrazit si statistky pro více stránek, šablona musela umět zobrazit pole. K těmto účelům jsou v FTL direktiva. Na místo, kde se měl zobrazit seznam stránek, jsem vložil direktiva pro kolekce a FTL se postaral o zbytek.

4.3.6 JavaScript pro smazání uzlů

Smazání uzlů nebylo možné udělat přes SQL dotaz, protože by dotaz překročil maximální dobu, po kterou se můžou dotazy vykonávat a tím by se provedl ROLLBACK dotazu. Kvůli tomu jsem se rozhodl použít metodu rekurzivního procházení uzlů a následné smazání. Pro průchod jednotlivými uzly jsem znova použil algoritmus Prohledávání do hloubky. Funkcionalita je v podstatě stejná jako u Web Scriptu využití stránek, rozdíl je v tom, že tento JavaScript kód se pouští pouze pro jednu stránku a uzly se začínají mazat, až algoritmus narazí na list.

4.3.7 Web Script pro změnu práv určité složky

Web Script může být spuštěn ve dvou módech, v módu pro čtení a v módu zápisu kdy se provede změna práv. Mód pro čtení slouží pro zjištění, jaká bude doba vykonání Web Scriptu, a které složky budou ovlivněny. Mód zadává uživatel jako parametr v URL adrese.

Uživatel dále určuje jméno stránky, pro kterou se má Web Script provést. Pokud není jméno zadáno nebo obsahuje nepovolené znaky, automaticky se berou všechny stránky.

Na začátku je sestaven SQL dotaz pro získání stránek na základě parametrů v URL adrese. Poté se zavolá funkce, do které se předají všichni přímí potomci stránky. Uvnitř funkce se procházejí potomci a hledá se složka documentLibrary, stránka nemusí tuto složku mít, pokud nemá žádné dokumenty nebo složky. Po nalezení documentLibrary se sestaví SQL dotaz, vyhledávající složku, pro kterou se mají změnit práva. Aby byl dotaz co nejefektivnější, používá funkci IN_FOLDER, která omezuje vyhledávání pouze na zadanou složku (v tomto případě documentLibrary). Pokud dotaz nevrátí složku, je vytvořen objekt oznamující, že nenašel složku. Pokud dotaz vrátí složku, jsou prohledána práva přiřazená této složce. Po nalezení práv pro skupinu EVERYONE jsou tato práva odstraněna a stejná práva jsou přidána pro interní skupinu. Nakonec se do objektu vloží přidaná a odstraněná práva a cesta na danou složku, pak je objekt přidán do pole. Toto pole spolu s dalšími věcmi (počet stránek, název stránky a jiné) je předáno do root objektu pro zobrazení ve FreeMarker šabloně.

Poté jsem vytvořil XML konfigurační soubor pro Web Script a následně jsem vytvořil FreeMarker šablonu, kde jsem vypisoval pole stránek, které Web Script procházel.

5 Závěr

5.1 Znalosti a dovednosti uplatněné v průběhu odborné praxe

Veškeré projekty, na kterých jsem pracoval, byly vyvíjeny v jazyce Java. Z toho důvodu jsem nejvíce znalostí využil z předmětu *Programovací jazyky I*. Při vývoji aplikací mi pomohly znalosti vývoje softwaru a návrhových vzorů z předmětů *Vývoj informačních systémů* a *Úvod do softwarového inženýrství*. Skoro po celou dobu odborné praxe jsem byl ve styku s SQL dotazy. Při těchto příležitostech jsem uplatňoval znalosti z předmětu *Úvod do databázových systémů*. U druhého projektu jsem využil hodně znalostí z předmětu *Databázové a informační systémy*, konkrétně vytvoření vlastního ORM.

5.2 Scházející znalosti a dovednosti v průběhu odborné praxe

Předmět *Programovací jazyky I* mi dal základy ohledně programovacího jazyka Java, ovšem při vývoji aplikací jsem používal například nejnovější verzi Javy a API pro práci s HTTP klientem, tyto znalosti mi scházely. Také jsem neměl zkušenost s objektově orientovaným jazykem JavaScript, který jsem využíval při práci na Tieto Share 2.0. Scházely mně znalosti platformy Alfresco a frameworků Spring, Hibernate a PrimeFaces. Jako nevýhodu považuji neznalost jakéhokoliv nástroje správy verzí kódu například Git.

5.3 Celkové zhodnocení odborné praxe a dosažených výsledků

Tuto praxi hodnotím jako jednu z nejcennějších částí mého studia. Byla to velice dobrá příležitost jak získat praktické zkušenosti ke zkušenostem teoretickým, které jsem získal během studia.

Během praxe jsem se podílel na vývoji interních projektů. Měl jsem tak možnost pracovat s novými a zajímavými technologiemi, jako jsou frameworky Spring a Hibernate nebo také platforma Alfresco. Také jsem se seznámil s agilní metodikou vývoje softwaru Scrum, kterou jsem provozoval v českém i anglickém jazyce. Díky spolupráci se zkušenými programátory jsem se naučil vytvářet lépe čitelný a udržovatelný kód.

V průběhu absolvování praxe jsem tak získal mnoho užitečných zkušeností, které budu moci využít nejen při dalším studiu, ale i v budoucím profesním životě.

Literatura

[1] Historie Tieta [online] [cit. 2016-04-5]. Dostupné z: http://www.tieto.cz/tieto-o-nas/historie-tieto-czech-republic

[2] Tieto o nas [online] [cit. 2016-04-5]. Dostupné z: http://www.tieto.cz/tieto-o-nas

[3] Spring Security Framework [online] [cit. 2016-04-5]. Dostupné z: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/introduction.html

[4] Alfresco ECM platforma [online] [cit. 2016-04-5]. Dostupné z: https://www.alfresco.com/products/enterprise-content-management

[5] Alfresco Web Scripts [online] [cit. 2016-04-5]. Dostupné z: https://wiki.alfresco.com/wiki/Web_Scripts

[6] Apache FreeMarker [online] [cit. 2016-04-5]. Dostupné z: http://freemarker.org/

[7] PrimeFaces Framework [online] [cit. 2016-04-5]. Dostupné z: http://primefaces.org/

[8] Anti-pattern The God Class [online] [cit. 2016-04-5]. Dostupné z: https://sourcemaking.com/antipatterns/the-blob

[9] Apache Chemistry OpenCMIS [online] [cit. 2016-04-5]. Dostupné z: https://chemistry.apache.org/java/developing/guide.html

[10] Alfresco AMP soubory [online] [cit. 2016-04-5]. Dostupné z: https://wiki.alfresco.com/wiki/AMP_Files